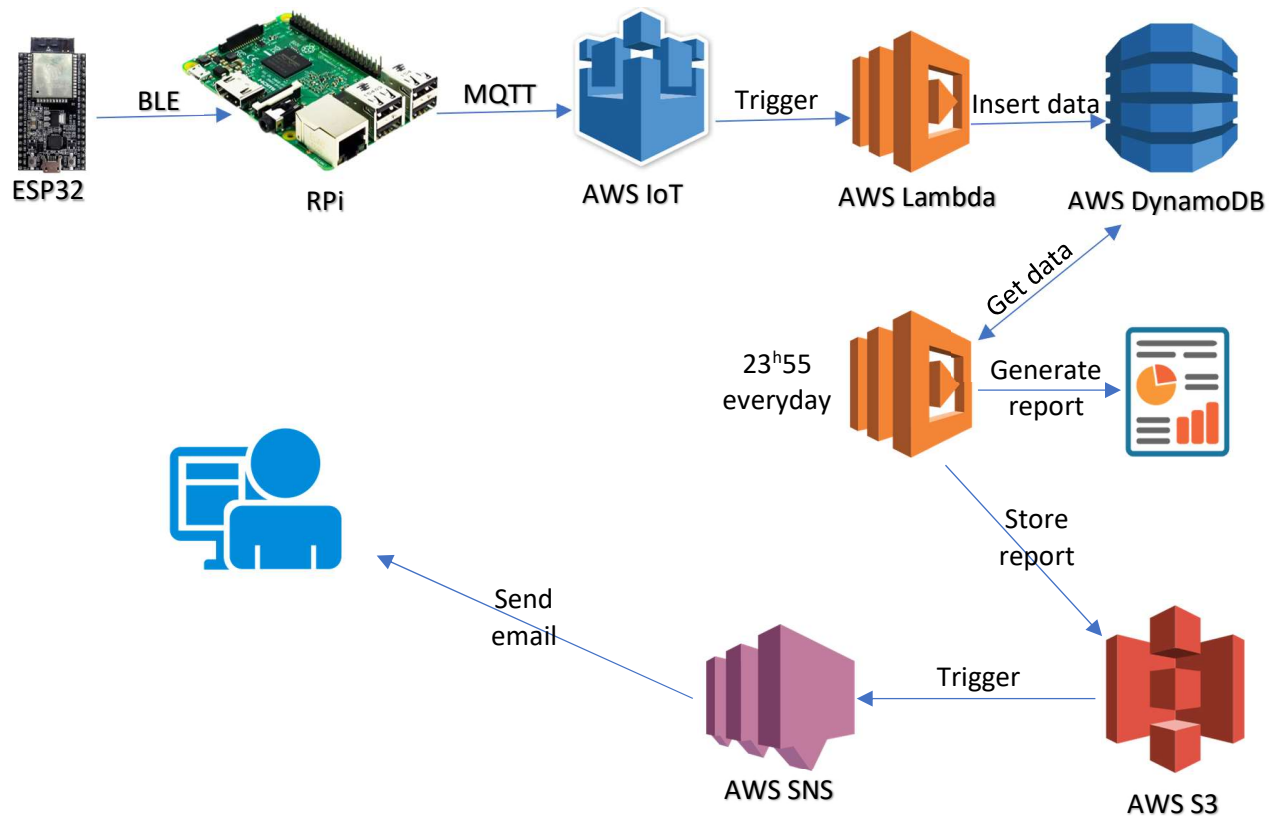


1. Server side
a. General process



b. RPi → AWS IoT (send data through MQTT)

RPi(s) publish MQTT messages on topic (e.g. RPi/today), so we can subscribe to the topic to view the messages published to it. (more details in Hardware side (Dat))

MQTT client [?](#) Connected as iotconsole-1499609523234-0

Subscriptions	RPi/today	Clear	Pause
Subscribe to a topic	RPi/today Jul 9, 2017 10:20:27 PM +0800		Hide
RPi/today	<pre>[{"MacAddress": "12:34:56:ab:cd:ef", "Serial": "101", "Count": "20", "Timestamp": "1499308864444"}, {"MacAddress": "12:34:56:ab:cd:ef", "Serial": "102", "Count": "21", "Timestamp": "1499308865798"}, {"MacAddress": "12:34:56:ab:cd:ef", "Serial": "103", "Count": "22", "Timestamp": "14993088667897"}, {"MacAddress": "12:34:56:ab:cd:ef", "Serial": "104", "Count": "23", "Timestamp": "1499308867979"}, {"MacAddress": "12:34:56:ab:cd:ef", "Serial": "105", "Count": "24", "Timestamp": "1499308868743"}]</pre>		Hide
	RPi/today Jul 9, 2017 10:19:50 PM +0800 <pre>[{"MacAddress": "12:34:56:ab:cd:ef", "Serial": "96", "Count": "15", "Timestamp": "1499308864444"}, {"MacAddress": "12:34:56:ab:cd:ef", "Serial": "97", "Count": "16", "Timestamp": "1499308865798"}, {"MacAddress": "12:34:56:ab:cd:ef", "Serial": "98", "Count": "17", "Timestamp": "14993088667897"}, {"MacAddress": "12:34:56:ab:cd:ef", "Serial": "99", "Count": "18", "Timestamp": "1499308867979"}, {"MacAddress": "12:34:56:ab:cd:ef", "Serial": "100", "Count": "19", "Timestamp": "1499308868743"}]</pre>		

c. AWS IoT → AWS DynamoDB

We create a rule for AWS IoT so whenever AWS IoT receive messages from RPi(s), a Lambda function will be executed.

The screenshot shows the AWS IoT console interface for a rule named 'sendBatchRule'. The rule is currently 'ENABLED'. The 'Overview' tab is selected. The 'Description' section shows 'No description'. The 'Rule query statement' section contains the SQL query 'SELECT * FROM 'RPi/today'' and notes it is using SQL version 2016-03-23. The 'Actions' section shows one action: 'Invoke a Lambda function passing the ...' with the function name 'sendBatchFunction'. There is an 'Add action' button at the bottom.

RULE
sendBatchRule
ENABLED Actions ▾

Overview Description Edit

No description

Rule query statement Edit


The source of the messages you want to process with this rule.

```
SELECT * FROM 'RPi/today'
```

Using SQL version 2016-03-23

Actions

Actions are what happens when a rule is triggered. [Learn more](#)

 **Invoke a Lambda function passing the ...** Remove Edit ▾
sendBatchFunction

Function name sendBatchFunction

[Add action](#)

The rule will take a message AWS IoT just receive to be a source for action, so we state a query:

“ SELECT * from ‘RPi/today’ ”

And add a Lambda function as an action.

AWS Lambda

Dashboard
Functions

Lambda > Functions > sendBatchFunction

ARN - arn:aws:lambda:ap-southeast-1:49810:

Qualifiers Test Actions

Code Configuration Triggers Tags Monitoring

Code entry type Edit code inline

```

1 from __future__ import print_function
2 import boto3
3 import json
4 import decimal
5 dynamodb = boto3.resource('dynamodb', region_name='ap-southeast-1', endpoint_url="http://dynamodb.ap-southeast-1.amazonaws.com")
6 table = dynamodb.Table('RPiCountData')
7
8 def lambda_handler(event, context):
9
10     for e in event:
11         rawData = json.dumps(e)
12         data = json.loads(rawData)
13         macAddress = data['MacAddress']
14         serial = data['Serial']
15         count = data['Count']
16         timestamp = data['Timestamp']
17
18         table.put_item(
19             Item={
20                 'MacAddress': macAddress,
21                 'Serial': serial,
22                 'Count': count,
23                 'Timestamp': timestamp
24             }
25         )

```

In the function, we connect to DynamoDB's table by:

```

5 dynamodb = boto3.resource('dynamodb', region_name='ap-southeast-1', endpoint_url="http://dynamodb.ap-southeast-1.amazonaws.com")
6 table = dynamodb.Table('RPiCountData')

```

Data in the message that we query above will be in “event” variable, so we convert it into array of JSON data and insert data into DynamoDB table.

```

8 def lambda_handler(event, context):
9     for e in event:
10         rawData = json.dumps(e)
11         data = json.loads(rawData)
12         macAddress = data['MacAddress']
13         serial = data['Serial']
14         count = data['Count']
15         timestamp = data['Timestamp']
16
17         #Insert data into DynamoDB table
18         table.put_item(
19             Item={
20                 'MacAddress': macAddress,
21                 'Serial': serial,
22                 'Count': count,
23                 'Timestamp': timestamp
24             }
25         )

```

We also must create a role and give that role permission (by attach policies) to execute the function.

The screenshot shows the AWS IAM console interface. On the left is a navigation sidebar with links to Dashboard, Groups, Users, Roles (highlighted), Policies, Identity providers, Account settings, Credential report, and Encryption keys. The main content area is titled 'IAM > Roles > helloworld-dev'. Below this is a 'Summary' tab showing the Role ARN (arn:aws:iam::498107424281:role/helloworld-dev), Role description, Instance Profile ARNs, Path (/), and Creation time (2017-04-23 11:02 UTC+0800). Below the summary are tabs for Permissions, Trust relationships, Access Advisor, and Revoke sessions. The 'Permissions' tab is active, showing a section 'Managed Policies' with a note: 'The following managed policies are attached to this role. You can attach up to 10 managed policies.' Below this is a table of attached policies.

Policy Name	Actions
AWSLambdaFullAccess	Show Policy Detach Policy Simulate Policy
AmazonDynamoDBFullAccess	Show Policy Detach Policy Simulate Policy
AWSLambdaDynamoDBExecutionRole	Show Policy Detach Policy Simulate Policy
AmazonDynamoDBReadOnlyAccess	Show Policy Detach Policy Simulate Policy
AWSLambdaExecute	Show Policy Detach Policy Simulate Policy
AWSLambdaReadOnlyAccess	Show Policy Detach Policy Simulate Policy
AWSLambdaInvocation-DynamoDB	Show Policy Detach Policy Simulate Policy

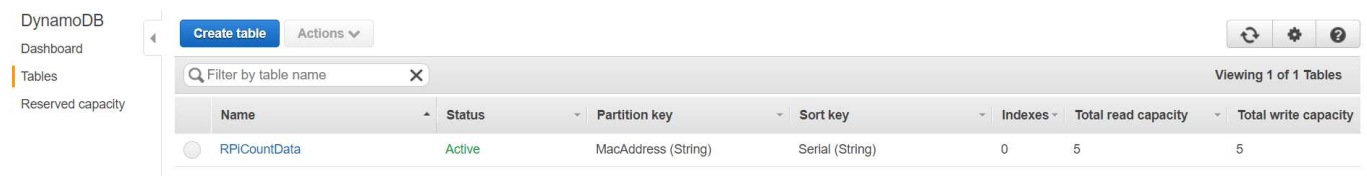
AWS Lambda

Dashboard
Functions

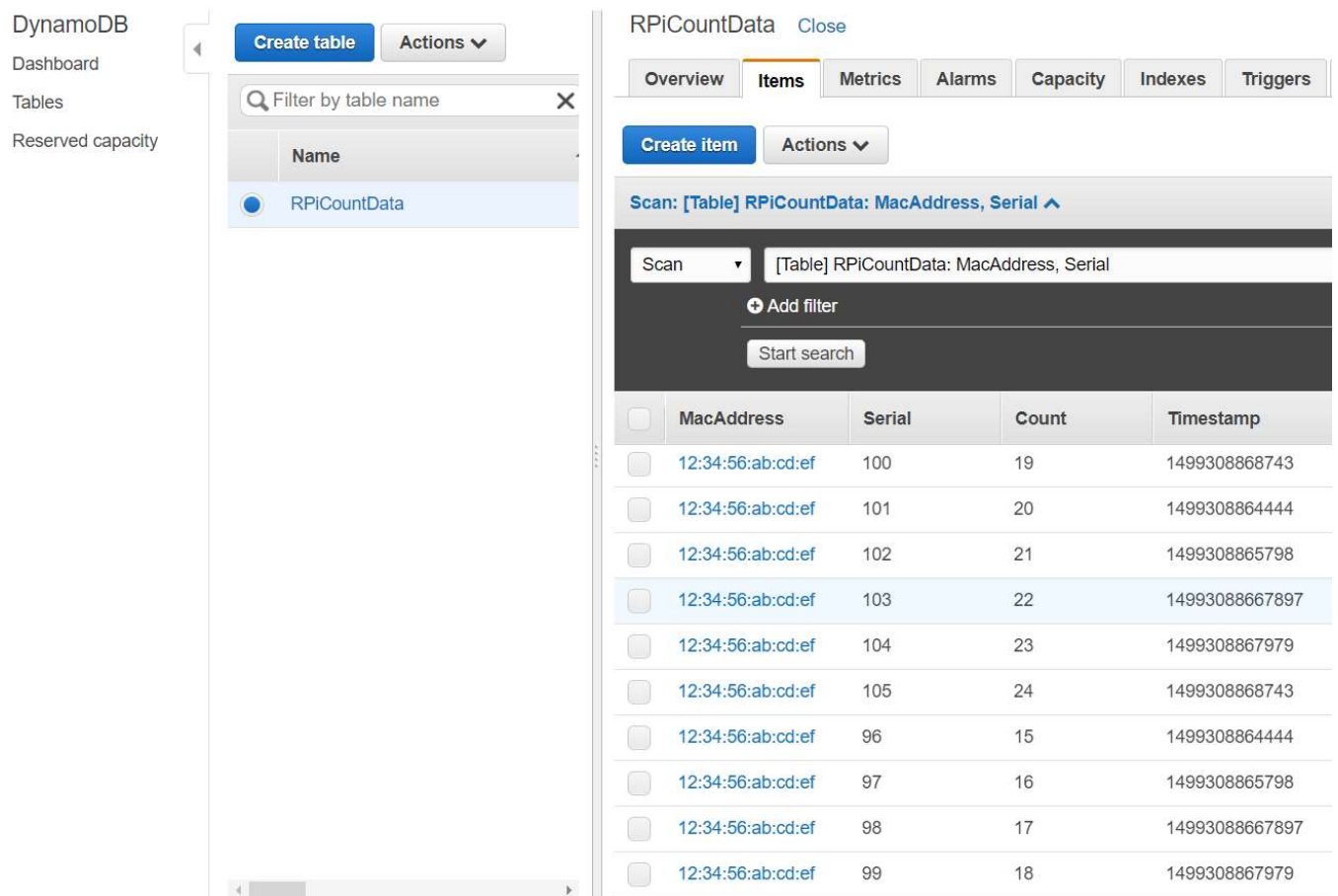
The screenshot shows the AWS Lambda console interface. The breadcrumb trail is 'Lambda > Functions > sendBatchFunction'. There are buttons for 'Qualifiers', 'Save', 'Save and test', and 'Actions'. Below these are tabs for 'Code', 'Configuration' (active), 'Triggers', 'Tags', and 'Monitoring'. The 'Configuration' tab shows the following settings:

- Runtime:** Python 3.6
- Handler:** lambda_function.lambda_handler
- Role:** Choose an existing role
- Existing role:** helloworld-dev

In DynamoDB, we create a table to store data.



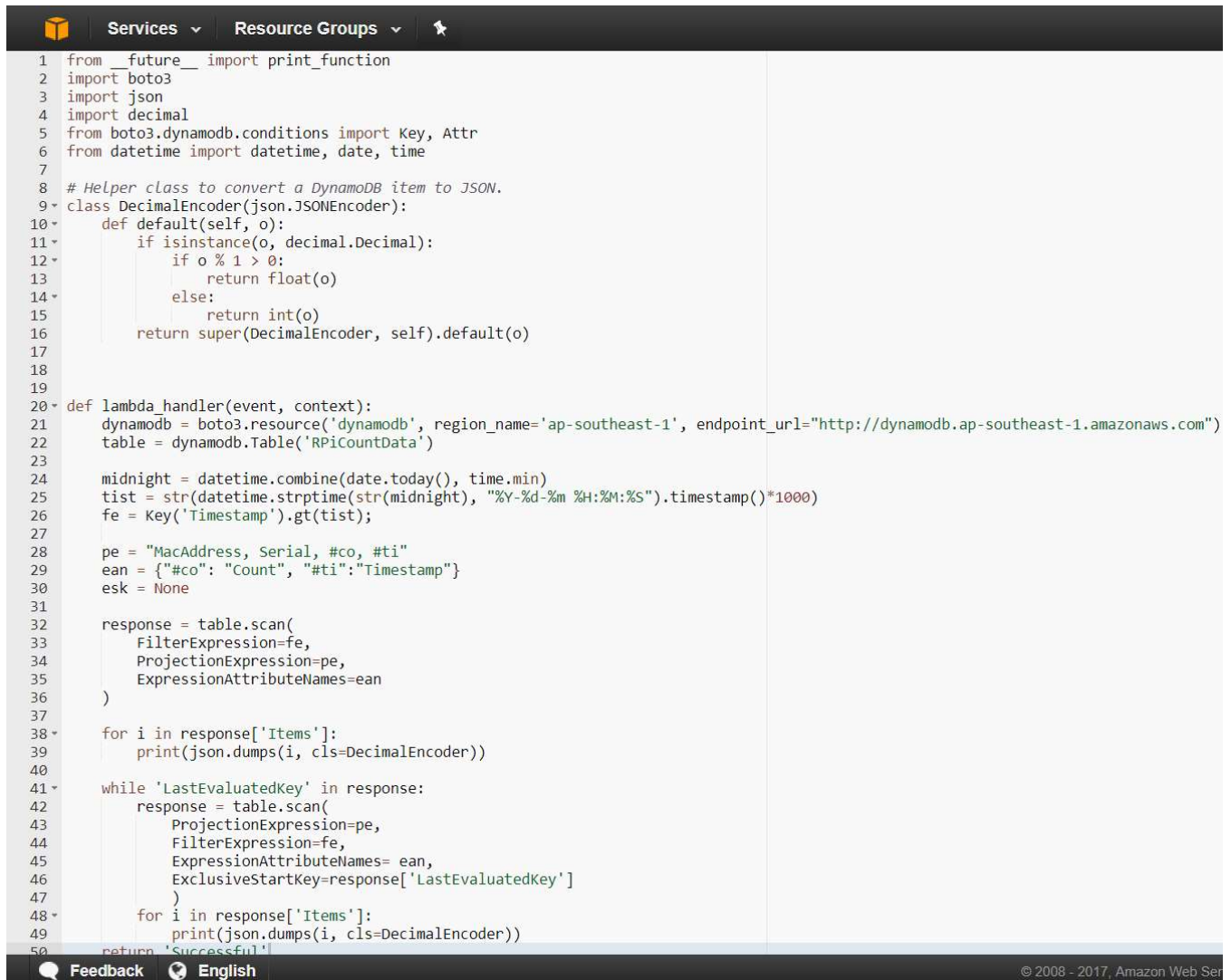
And then whenever RPi(s) send message to AWS IoT, data will be insert into DynamoDB table:



So, all the process will be:

- Create a AWS role, attach policies.
- Create a DynamoDB table.
- Create a Lambda function to process incoming data from RPi and update them to DynamoDB table, configure function's role with role created above.
- Subscribe to MQTT client topic to view incoming data from RPi.
- Create a rule so whenever a MQTT topic have new coming message, it will excute the Lambda function above.

- d. Create a Lambda function to get data from DynamoDB's table and generate report at the end of the day



```

1  from __future__ import print_function
2  import boto3
3  import json
4  import decimal
5  from boto3.dynamodb.conditions import Key, Attr
6  from datetime import datetime, date, time
7
8  # Helper class to convert a DynamoDB item to JSON.
9  class DecimalEncoder(json.JSONEncoder):
10     def default(self, o):
11         if isinstance(o, decimal.Decimal):
12             if o % 1 > 0:
13                 return float(o)
14             else:
15                 return int(o)
16         return super(DecimalEncoder, self).default(o)
17
18
19
20 def lambda_handler(event, context):
21     dynamodb = boto3.resource('dynamodb', region_name='ap-southeast-1', endpoint_url="http://dynamodb.ap-southeast-1.amazonaws.com")
22     table = dynamodb.Table('RPiCountData')
23
24     midnight = datetime.combine(date.today(), time.min)
25     tist = str(datetime.strptime(str(midnight), "%Y-%d-%m %H:%M:%S").timestamp()*1000)
26     fe = Key('Timestamp').gt(tist);
27
28     pe = "MacAddress, Serial, #co, #ti"
29     ean = {"#co": "Count", "#ti": "Timestamp"}
30     esk = None
31
32     response = table.scan(
33         FilterExpression=fe,
34         ProjectionExpression=pe,
35         ExpressionAttributeNames=ean
36     )
37
38     for i in response['Items']:
39         print(json.dumps(i, cls=DecimalEncoder))
40
41     while 'LastEvaluatedKey' in response:
42         response = table.scan(
43             ProjectionExpression=pe,
44             FilterExpression=fe,
45             ExpressionAttributeNames= ean,
46             ExclusiveStartKey=response['LastEvaluatedKey']
47         )
48     for i in response['Items']:
49         print(json.dumps(i, cls=DecimalEncoder))
50     return 'Successful'

```

Feedback English © 2008 - 2017, Amazon Web Ser

For easier process data get from DynamoDB's table, we convert it into JSON:

```

8  # Helper class to convert a DynamoDB item to JSON.
9  class DecimalEncoder(json.JSONEncoder):
10     def default(self, o):
11         if isinstance(o, decimal.Decimal):
12             if o % 1 > 0:
13                 return float(o)
14             else:
15                 return int(o)
16         return super(DecimalEncoder, self).default(o)
17

```

Connect to DynamoDB's table:

```
21 dynamodb = boto3.resource('dynamodb', region_name='ap-southeast-1', endpoint_url="http://dynamodb.ap-southeast-1.amazonaws.com")
22 table = dynamodb.Table('RPICountData')
23
```

To get all data of the last day, we create a midnight timestamp of the day before and compare it with timestamp field of all data, and just get data satisfied (greater than last midnight timestamp):

```
24 midnight = datetime.combine(date.today(), time.min)
25 tist = str(datetime.strptime(str(midnight), "%Y-%d-%m %H:%M:%S").timestamp()*1000)
26 fe = Key('Timestamp').gt(tist);
27
28 pe = "MacAddress, Serial, #co, #ti"
29 ean = {"#co": "Count", "#ti": "Timestamp"}
30 esk = None
31
32 response = table.scan(
33     FilterExpression=fe,
34     ProjectionExpression=pe,
35     ExpressionAttributeNames=ean
36 )
37
```

ProjectionExpression (pe) specifies the attributes we want in the scan result.

FilterExpression (fe) specifies a condition that returns only items that satisfy the condition. All other items are discarded.

ExpressionAttributeNames (ean) provides name substitution, we use this because “count” and “timestamp” is a reserved word in DynamoDB, we cannot use it directly in any expression, so we use the expression attribute name “#co” and “#ti” to address this.

The scan method returns a subset of the the items each time, called a page. The LastEvaluatedKey value in the response is then passed to the scan method via the ExclusiveStartKey parameter. When the last page is returned, LastEvaluatedKey is not part of the response:

```
41 while 'LastEvaluatedKey' in response:
42     response = table.scan(
43         ProjectionExpression=pe,
44         FilterExpression=fe,
45         ExpressionAttributeNames= ean,
46         ExclusiveStartKey=response['LastEvaluatedKey']
47     )
48 for i in response['Items']:
49     print(json.dumps(i, cls=DecimalEncoder))
```

All data we get now in “response[‘Items’]” as JSON and we can generate report from it.

We have not implemented code to generate report yet due to format problem.

To schedule the Lambda function executed at 23^h55 daily, we use AWS CloudWatch to do it

Rules > ScheduleGenerateReport

Summary

ARN ⓘ `arn:aws:events:ap-southeast-1:498107424281:rule/ScheduleGenerateReport`

Schedule Cron expression `55 23 * * * *`

Next 10 Trigger Date(s)

1. Sun, 09 Jul 2017 23:55:00 GMT
2. Mon, 10 Jul 2017 23:55:00 GMT
3. Tue, 11 Jul 2017 23:55:00 GMT
4. Wed, 12 Jul 2017 23:55:00 GMT
5. Thu, 13 Jul 2017 23:55:00 GMT
6. Fri, 14 Jul 2017 23:55:00 GMT
7. Sat, 15 Jul 2017 23:55:00 GMT
8. Sun, 16 Jul 2017 23:55:00 GMT
9. Mon, 17 Jul 2017 23:55:00 GMT
10. Tue, 18 Jul 2017 23:55:00 GMT

Status Enabled

Description Schedule generate report of using Sanitizers

Monitoring [Show metrics for the rule](#)

Targets

Filter:

Type	Resource name	Input	Role	Additional parameters
Lambda function	generateReport	Matched event		

About the report, we can think about some questions like:

- Which time of day the sanitizers are used most?
- Which venue the sanitizers are used most?
- Which day the sanitizers are used most?
- List of all sanitizers which is less than 30% of volume.

...